ESD-TR-85-293

(2)

Semiannual Technical Summary

Restructurable VLSI Program

30 September 1985

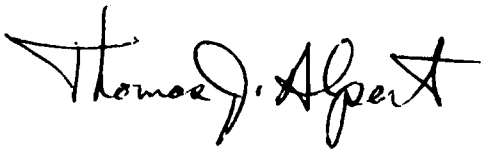
Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

Prepared for the Defense Advanced Research Projects Agency
under Electronic Systems Division Contract F19628-85-C-0002.

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Defense Advanced Research Projects Agency under Air Force Contract F19628-85-C-0002 (ARPA Order 3797).

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



Thomas J. Alpert, Major, USAF
Chief, ESD Lincoln Laboratory Project Office

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

RESTRUCTURABLE VLSI PROGRAM

SEMIANNUAL TECHNICAL SUMMARY REPORT
TO THE
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

1 APRIL — 30 SEPTEMBER 1985

ISSUED 14 JANUARY 1986

LEXINGTON MASSACHUSETTS

ABSTRACT

This report describes work performed on the Restructurable VLSI Program sponsored by the Information Processing Techniques Office of the Defense Advanced Research Projects Agency during the period 1 April through 30 September 1985.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

RESTRUCTURABLE VLSI PROGRAM

I. PROGRAM OVERVIEW AND SUMMARY

A. OVERVIEW

The main objective of the Lincoln Restructurable VLSI (RVLSI) Program is to develop design methodologies, architectures, design aids, and testing strategies for implementing wafer-scale systems with complexities approaching a million gates. In our approach, we envisage a modular style of architecture comprising an array of cells embedded in a regular interconnection. Ideally, the cells should consist of only a few basic types. The interconnection matrix is a fixed pattern of metal lines augmented by a complement of programmable switches or links. Conceptually, the links could be either volatile or nonvolatile. They could be of an electronic nature, such as a transistor switch, or could be permanently programmed through some mechanism such as a laser. The RVLSI Program is currently focusing on laser-formed interconnect.

The link concept offers the potential for a highly flexible, restructurable type of interconnect technology that could be exploited in a variety of ways. For example, logical cells or subsystems found to be faulty at wafer-probe time could be permanently excised from the rest of the wafer. The flexible interconnect also could be used to circumvent faulty logic and tie in redundant cells judiciously scattered around the wafer for this purpose. Also, the interconnect could be tailored to a specific application in order to minimize electrical degradations and performance penalties caused by unused wiring and links.

Further, the testing of a particular logical subsystem buried deep within a complex wafer-scale system poses a very difficult problem. A properly designed restructurable interconnect matrix could be temporarily configured to improve both the controllability and observability of internal cells from the wafer periphery. In this way, each component cell or a manageable cluster of cells could be tested in a straightforward manner using standard techniques. With an electronic linking mechanism, it is possible to think in terms of a dynamically reconfigurable system. Such a feature could be used to alter the function mode of a system to changes in the operating scenario, or it could be used to support some degree of fault tolerance if the system architecture was suitably designed.

Several major areas of research have been identified in the context of the RVLSI concept:

(1) System architectures and partitioning for whole-wafer implementation.

(2) Placement and routing strategies for optimal utilization of redundant resources and efficient interconnect.

(3) Assignment and linking algorithms to exploit redundancy and flexible interconnect.

(4) Methods for expediting cell design with emphasis on functional level descriptions, enhanced testability, and fault tolerance.

(5) Methods for testing complex, multiple-cell, whole-wafer systems.

Complementary work on the development of various link interconnect technologies as well as fabrication/processing technology is being supported by the Lincoln Air Force Line Program, and results are reported under the Lincoln Laboratory Advanced Electronic Technology Quarterly Technical Summary.

B. SUMMARY OF PROGRESS

Work for this period is reported under two headings: Design Aids for RVLSI (Section II) and Applications (Section III).

1. Design Aids for RVLSI

Design has started on a new version of the wafer design and linking tools. The Standard Linking Automation Shell (SLASH) will include a number of new features such as automatic wafer floor planning and incremental restructuring.

The Phase I DSP Silicon Compiler is complete. The first sample chips are being fabricated in MOSIS and will be tested shortly.

The design of the architecture and algorithm for the Phase II compiler is complete. Work is progressing on the coding of the algorithms and the layout of the necessary cells.

2. Applications

The construction of the Dynamic Time Warping (DTW) wafer system for speech recognition is progressing. The design of the wafer itself is complete. Each of the two cells has been fabricated in MOSIS on discrete chips. These chips have been tested and corrected versions have been submitted. The hardware and software in the wafer-support system is complete and tested to the extent possible without either the wafer or a set of discrete chips.

II. DESIGN AIDS

A. LSH

We have begun the process of building SLASH (Standard Linking Automation Shell), a new RVLSI CAD tool to replace LSH. SLASH will include features that cannot be implemented by enhancing LSH. These include, for example, floor planning and incremental restructuring. The floor-planning feature will permit production of a physical wafer description from a logical description consisting of a functional block/netlist description of the logical system, area estimates for the functional blocks, the design rules, and the yield statistics. Incremental restructuring will permit restructuring and testing a wafer-scale system beginning with a small subsystem, restructuring, testing, augmenting that small subsystem, restructuring, testing, augmenting again, and so forth. We have produced initial SLASH specifications for the programming environment and for the initial functions to be built, and are now in the early stages of system implementation.

While the SLASH system is in its early stages, we will continue the parallel activity of enhancing LSH to meet current needs. For example, we have added a new wire routing program to LSH, in anticipation of the CAD tool needs of the RVLSI Dynamic Time Warping project. This routing program was designed to be optimal for "subcircuit" wire routing problems. For example, given a circuit on a wafer consisting of a chain of congruent subcircuits connected in series, and given another subcircuit on the wafer which is not connected to the chain, the program will find the most efficient reconfiguration procedure to add this new subcircuit to the chain.

B. DSP SILICON COMPILER

1. Phase I

a. Status

The Phase I compiler has been completed and is now in operation. As previously reported, the compiler is able to process fixed-topology filters. The technology is the two-level metal, 3-μm bulk CMOS currently supported by MOSIS.

Several designs have been submitted to MOSIS for fabrication. Unfortunately the fabrication run has been significantly delayed, so we have not received any chips to test.

b. Operation

The compiler operates in two independent steps. First, the filter descriptions indicated in the high-level input language are converted into a detailed netlist that contains the required hardware resources and their interconnect, including all the control signals for the arithmetic units and any delays necessary to compensate for unit latencies. A human-readable file with this description is produced as output of this first phase. Then, the layout program takes this intermediate description and produces the complete layout, including the input/output pads and power and clock distribution.

c. Layout Generation

The chips produced by the Phase I compiler are assembled using a library of basic cells. Initially, each larger hardware unit is built using the basic cells, and then these larger units are placed on the chip according to a fixed floor plan.

Presently, the basic hardware units comprise adders, subtracters, arbitrary length shift registers, and fixed coefficient multipliers. Each multiplier is specifically generated based on the coefficient. Shift registers are produced according to length in a straight line, U-shape, or double U to avoid extreme aspect ratios. As the main control signal for the arithmetic units is the least-significant bit (LSB) of the data, the LSB of the result is provided as one output by all arithmetic units.

The technology used to implement the circuits is the current MOSIS two-level metal, 3-μm bulk CMOS. The second level of metal is being used only for power distribution.

The floor plan consists of two vertical columns of units and a routing channel between them. The right side contains the multipliers and the left side the rest of the units. Clocks are distributed to the columns from the side opposite to the central channel. Connections between the columns and to the pads go through the channel, routed by a simple channel router that uses the polysilicon and first-level metal layers. The input/output signals are routed to the pads from the top and bottom of the channel by a river router on first-level metal.

The entire layout section of the Phase I compiler is written in LISP using the L5 Layout Language* previously developed at Lincoln Laboratory. The code comprises about 4000 lines, excluding the library items and the L5 package.

The library cells were originally designed using the University of California Caesar Graphic Editor and converted to L5 format. Other CAD tools employed for checking and simulations are Lyra (geometrical design rule checks), SPICE (circuit simulations), and RNL (switch-level simulation).†

We found it useful to compare simulation results by means of programs that automatically converted input and output formats between signal format and the serial bit-level description required by RNL. Several special conversion routines were written for this purpose, making it much easier to check different types of simulations against each other.

d. Circuits Submitted to MOSIS for Fabrication

Three designs have been submitted to MOSIS for fabrication. They contain IIR (first- and second-order recursive sections) and FIR (three tap) filters. These designs will be the first test for the Phase I DSP compiler.

---

* K.W. Crouch, "L5 User's Guide," Project Report RVLSI-5, Lincoln Laboratory, M.I.T. (March 1984); available upon request from Lincoln Laboratory.

† C.J. Terman, "User's Guide to NET, PRESIM, and RNL/NL — Expanded Version 3.3," M.I.T. Laboratory for Computer Science (September 1982).

2. Phase II

a. Source Language and Parser

The Phase II Silicon Compiler will allow time multiplexing of chip resources and will permit more general constructions in the input language. The design of the high-level source language appropriate to Phase II has been completed. The main features of this language can be understood by study of a few illustrative examples. Figure 1 shows the specification for a cascade of four first-order filter sections. The first part of the specification declares the data word size and the filter coefficient size. The next two statements declare the number of multipliers and adders available to the compiler for designing the chip. The compiled chip will always implement the specified function. However, it will run slower if fewer resources are available. If this declaration is not made, a default value of one is assumed. Next, pins 5 and 10 are declared as input and output labeled x and w, respectively. Pins 7, 8, and 9 carry bit synchronization input and output signals. The next statement declares z and q as internal variables of array and scalar type, respectively. Finally, e and f are declared as coefficient arrays each containing four coefficients. The size of the array variable z is not declared explicitly but is determined later when it is used in a loop context.

```
wordsize 28;
coefsize 12;

nadd 2;
nmult 2;

input x 5;
output w 10;

input lsb_in 7;
output lsb_out 8;
output lsb_strobe 9;

var z[], q;

coef e[] = 85, -98, 54, 76;
coef f[] = 85, -98, 54, 76;

%

loop(q <- x(#), 4, w <- q) {
    z[](#) = e[] * z[](#-1) + q;
    q = f[] * z[](#-1) + z[](#);
}
```

Figure 1. Cascade filter specification.

158022-N

The remaining part of the filter specification consists mainly of two algebraic expressions defining a single first-order section. These expressions are to be iterated four times in a loop. The array variables and coefficients appearing in this loop do not require indexes; they are assumed to start at 0 and increment once per iteration of the loop. The pre-assignment q ← x(#) appearing in the loop statement means that q is to be assigned the value of x(#) **before** being used on the first iteration of the loop. Similarly, the post-assignment w ← q means that w is to be assigned the value of q **after** the last iteration of the loop. Pre- and post-assignments need not be present in the loop statement.

A somewhat more involved example of a high-level filter specification is shown in Figure 2. Here, y is declared as a time-multiplexed output pin, which means that successive assignments to y will be buffered as needed and output in sequence on that pin. The variable z and the coefficients e and f are declared as two-dimensional arrays. The sizes of the components of these arrays will be determined from the context in which they are later used. The body of the filter specification consists of two nested loops defining a bank of three filters, each of which is a cascade of two first-order sections. From the iteration length of these loops, it is determined that z, e, and f are [3,2] arrays. Should any of these arrays be used again in another nested loop, the new loop iteration lengths must agree with those of the earlier loops in which the array first appeared.

Figure 2. Simple filter bank specification.

```
wordsize 28,
coefsize 12,

input x 5,
muxout y 6,

input lsb_in 7,
output lsb_out 8,
output lsb_strobe 9,

var z[,], q,

coef e[,] = 85, -98, 54, 76, 632, -87,
coef f[,] = 85, -98, 54, 76, 76, -54,

%

loop(, 3, ) {
    loop(q <- x, 2, y <- q){
        z[,] (#) = e[,] * z[,] (#-1) + q,
        q = f[,] * z[,] (#-1) + z[,] (#);
    }
}
```

168023-N

As a final example of the language, Figure 3 gives the specification of the complete channel vocoder analyzer filter bank shown in block diagram form in Figure 4. There are 23 channels in the filter bank each of which consists of the cascade of two second-order sections followed by a full-wave rectifier followed by a first-order section. This output is downsampled by a factor of 4 and then passed through a second-order section before being sent off-chip as an output.

The only new features present in the declaration section of Figure 3 are the declaration of the variables t and z. (The use of ellipsis in the description is not a feature of the language but merely an editorial device to simplify the appearance of the figure.) The declaration of t establishes it as a time-multiplexed input array variable. This array is double buffered; a new set of array values is being read in while the last complete set read is being processed. The declaration of z establishes it as a multi-pin output array. This is again double buffered; the array values are stored as they are produced, while the last processed array is being read out with each individual array variable being assigned its own output pin.

The remaining part of the specification contains nothing new except the use of the full-wave rectification operator |x|. The first pair of nested loops implements that part of the block diagram in Figure 4 to the left of the downsampling outputs. The final loop takes care of the remaining second-order sections.

The Phase I parser has been completely rewritten to accommodate the language changes that are an integral part of Phase II of the silicon compiler project. The output of the parser is a parse tree similar to the one used in Phase I but having provision for the incorporation of nested loops. The parse tree contains, in a convenient form, all the information for the next stage of the *compilation process — scheduling.*

b. Algorithms and Architecture

The architecture of the Phase II machine remains much as it was described in a previous report. The augmentation of the language described in the preceding section impacts the design algorithms rather than the architecture.

As a result of doing several hand examples and reevaluating control needs, a new design of the control unit logic has been completed. This logic, shown in Figure 5, occupies a much smaller silicon area than the previous design. The control unit includes a micromemory for storing control words as well as the control unit logic. Each control word in micromemory contains four fields:

(1) A field for controlling bus switches, thus enabling hardware sharing.

(2) A field giving addresses for ROMs, which store state and intermediate variables.

(3) A 3-bit sequencing field, which gives commands to the control unit logic (this field is shown entering the "μ-control logic" box in Figure 5).

(4) A next address field, which is used primarily for storing branch addresses but may also store an initial loop count (this field is shown entering the "shift register" box in Figure 5).

```
wordsize 16;
coefsize 12;


nmult 2;          /* default value = 1 */
nadd 4;


muxin t[] 1;
output out[] 2, 3, ... , 24;
input lsb_in 25;
output lsb_out 26;
output lsb_strobe 27;


var x, u[], v[], w[], q[], z;


coef a[] = .575, ... ;
coef b[] = .675, ... ;
coef c[] = .775, ... ;
coef d[] = .875, ... ;
coef e[] = .975, ... ;
coef f[] = .474, ... ;
coef g[] = .375, ... ;


%


loop(;4;){
        x(#) = t[];
        loop(;23;){
                u[] (#) = a[] *u[] (#-1) - b[] *u[] (#-2) + x(#) - x(#-2);
                v[] (#) = c[] *v[] (#-1) - d[] *v[] (#-2)
                        + u[] (#) - u[] (#-2);
                z = |v[] (#)|;
                w[] (#) = e[] * (w[] (#-1) - z) + z;
        }
}
loop(;23;) {
        q[] (#) = f[] * (q[] (#-1) - w[] (#))
                - g[] * (q[] (#-2) - w[] (#)) + w[] (#);
        out[] (#) = q[] (#);
}
```

158024-N

Figure 3. Vocoder filter bank specification.

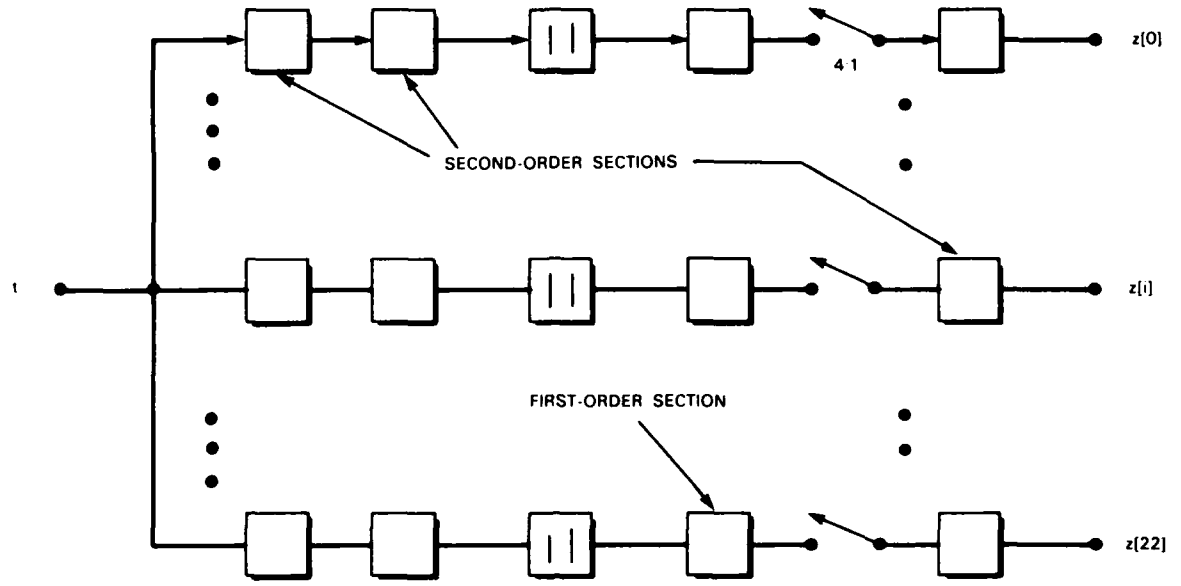


Figure 4. Channel vocoder filter bank.

The two main functions of the control unit logic are to generate a new address for micromemory each data word time (e.g., every 16 bits for a 16-bit data word) and to count loop iterations. Since the control unit logic has a full data word time to perform these functions, operations are performed bit serially with the implicit assumption that the length of the micromemory address is always less than the length of the data word. For realistic examples, this does not seem to pose any problem.

The "shift register" box in Figure 5 normally contains the program counter for micromemory When no program branch is required, the program counter is shifted into the "inc logic" box (a one-bit incrementer and associated multiplexing). If branching is required, the next address field is loaded into the shift register and a NOP operation is performed by the "inc logic" box. The MAR register holds the current micromemory address while a new one is being computed in the control unit logic.

The "dec logic" box decrements the loop count when loops are involved. The loop counts are stored in the memory files and are handled as other data. The "μ-control logic" box controls the operation of the control unit by decoding the sequencing instruction. The control unit logic is also required to perform other, sometimes relatively complex, functions involving nested loops and initialization of memories.

The removal from the language of the parallel construct "PAR" and the addition of resource declarations go hand-in-hand with a modification of the scheduling algorithm. The objective of the new scheduling algorithm is to construct the shortest schedule possible using only the resources declared by the user. The "PAR" construct is not necessary since the algorithm will always attempt to achieve maximum parallelism in use of available functional units while maintaining

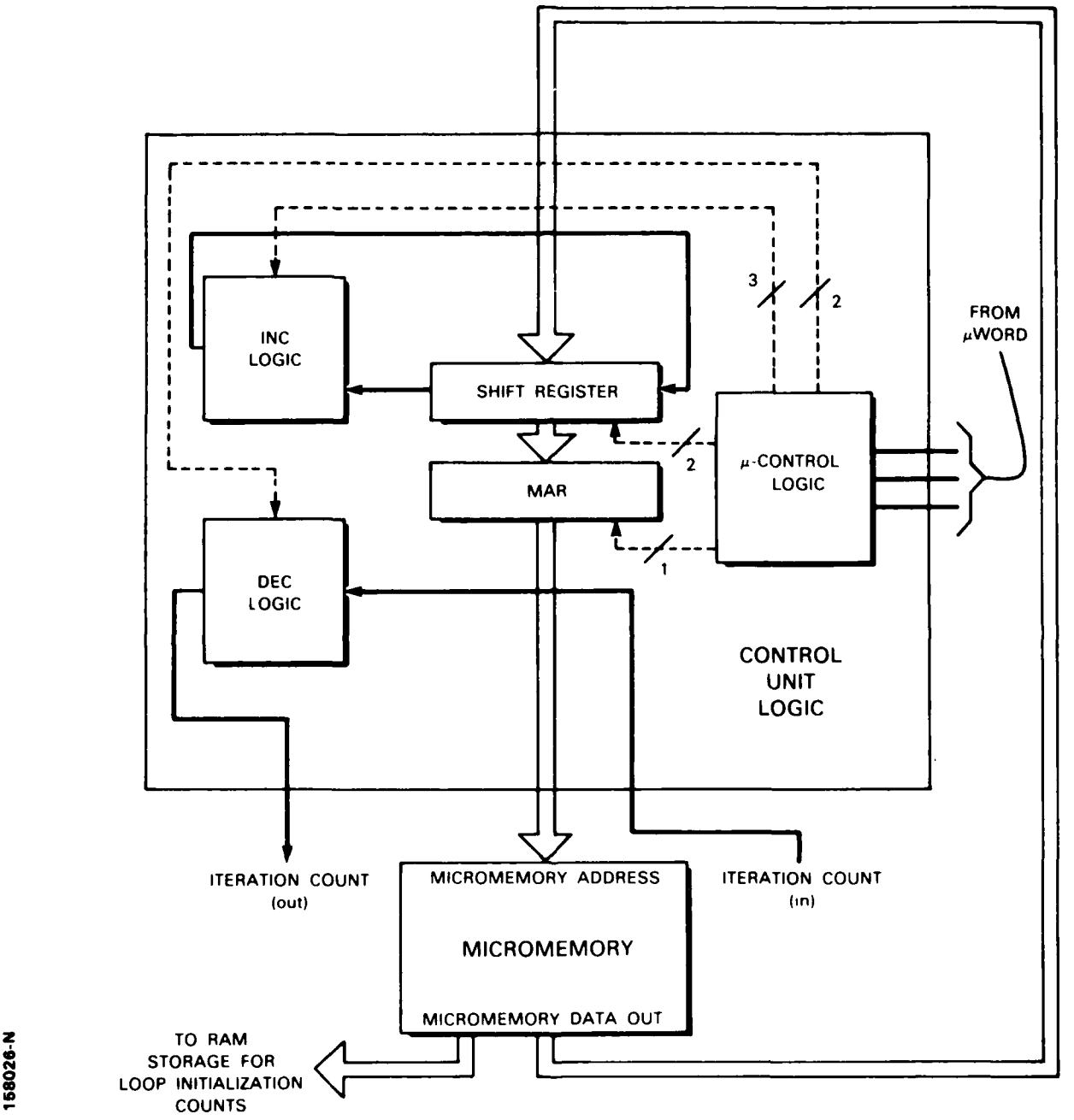


Figure 5. Control unit.

the behavior described by the user in the language. This approach has the additional advantage that the user may evaluate the effects of varying hardware resources without changing the behavioral description.

Algorithms to assign scheduled generic functional unit uses to actual functional units and to automatically generate microcode have been designed.

The algorithm for file and register assignment, i.e., assigning uses of files determined during scheduling to actual files and minimizing the number of registers in those files, has been programmed and is in the latter stages of debugging.

c. Cell Design

We have concentrated first on memory design for the Phase II compiler. The bit serial architecture of the compiler requires independent reads and writes and special input/output structures. To write into memory, the serial data must first go into a shift register from which it is written in parallel into the memory cells. A similar structure is required for the read operation.

The memory cell consists of three transistors: two of them connect the cell to the I/O lines, and the gate of the third is used to store the value. The time before a cell is written again is quite short, so there is no need for refreshing.

The output circuit consists of a sense amplifier, a shift register and multiplexers; the input circuit is similar but simpler. Design of all these circuits, including layouts, is now complete.

Several kinds of addressing circuits will be needed for the memories. We are presently designing these parts.

III. APPLICATIONS

A. DTW WAFER SYSTEM

1. Cell Design

The final DTW wafer-scale partitioning resulted in two cell types: a DCALC cell to perform a local distance calculation (speech frame to speech frame distance), and a PCALC cell to perform a path distance calculation (word-to-word comparison). To realize a wafer-scale processor to perform the speech recognition task, 42 of each of these processors shall be configured into a linear array of PCALC and DCALC pairs. At this time, layout of both PCALC and DCALC cells has been completed as well as the incorporation of these cell layouts into an interconnect matrix for the wafer-scale design.

The DCALC cell layout was the first to be completed. This cell layout consists of 3852 transistors and occupies an area of 2.2×2.7 mm, including I/O drivers and probing pads. The PCALC cell, also complete, contains 4657 transistors and occupies an area of 2.5×2.7 mm. These cells were submitted individually to the MOSIS foundry for logic verification in the July and August MOSIS CMOS runs. The cells were received and tested. Corrected designs have been submitted.

2. Wafer Design

The wafer-scale design has also been completed. It incorporates the existing PCALC and DCALC designs into the orthogonal two-level metal interconnection matrix. The wafer-scale design places side-by-side pairs of the PCALC and DCALC cells into an array 6 wide by 14 high. This puts 84 of each of the cells on the wafer, providing a 2:1 redundancy over the required 42 pairs. This array is bordered on all sides by I/O drivers to communicate data and control signals on and off the wafer system. Broadcast data and control are brought into the system on the top and bottom of the wafer while nearest-neighbor communication signals are terminated on the left and right sides. The linear array of PCALC/DCALC pairs is then defined by snaking the near-neighbor association alternately up and down the columns, proceeding from the left to the right on the wafer.

One of the major features of this design is the inclusion of cell bypass logic to aid in determining system functionality during the restructuring process. Included in each DCALC/PCALC logic cell is a single bit-shift register which has both its input and output available to restructurable wiring ports. This single bit will act as an ENABLE switch, so that if this bit is turned off, the logic of the DCALC/PCALC pair is disabled and all inputs are shunted to appropriate outputs. Conversely, when the bit is turned on, the cell will act on the data at the input ports and pass processed data to the outputs. Therefore, the ENABLE shift register can be used to configure the linear array so that any cell or any subset of the already connected cells can be tested in isolation. This provides a complete system testing and diagnostic capability from the external pins of the wafer at all times during restructuring.

The process of verifying and characterizing the cell designs is now under way. When this process is complete, any necessary corrections to the design will occur before submission of the wafer-scale design to the MOSIS foundry.

3. Support System

a. Software

The software for the DTW support system is being developed in three separate software packages: the operating system, DTW subsystem control, and front-end processing. The operating system continues to go through some refinements for better performance in a real-time environment, though the basic system is complete and operational. The DTW subsystem software is complete and undergoing some final debugging to correct a problem with word-to-byte DMA data transfers over the Multibus. The test package for the front end is now complete and working. The software for interfacing with the speech recognition system will be developed next.

The operating system consists of two main parts: a process scheduler and an I/O manager. It also includes utilities for string handling and conversions between integers and strings.

The scheduling of processes is preemptive and is based upon a simple priority-scheduling algorithm, thereby permitting several jobs to be executing on the system in a time-sharing fashion. This allows the front end to operate as a completely separate entity rather than as an integral part of the DTW speech recognition software. The two processes can share a defined section of data memory, including status information and the parameterized speech.

The I/O manager coordinates the requests for system resources such as the RS-232 ports, the DMA controller, and any peripherals added to the system. A collection of communication utilities allows the user to access the devices controlled by the I/O manager. Most of these utilities (i.e., read, write, getc) imitate the I/O capabilities of the same utilities used in "C." The system is general enough that the inclusion of a new resource requires only the addition of a device description and possibly a set of device drivers if special formats are required.

Many of the utilities included in the standard "C" library have been included in this operating system. All the utilities involving string handling (i.e., strcat, strlen, etc.) are available under the same formats as those in that library. Some conversion routines such as "itoa" and "ftoa" have been included as well as those for converting strings to integers and longs (atoi and atol). Dynamic memory allocation is handled by the "malloc," "calloc," and "free" commands just as in the standard "C" library. This allocated memory is automatically cleaned up for the user upon exiting program execution.

Programs written for execution by this system are written in "C" using the available utilities and linked to the operating system for execution. Each program is entered into a command interpreter list for identification. Arguments are passed to the programs using the same argc and argv arguments as in standard "C" programs.

A technique for storing user object code on a host system is being considered to handle the increasing program size which has just exceeded the available ROM storage space on the support system CPU board. The file handling capability will be primitive for easy implementation and may thereby require a more intimate knowledge of the program space by the user. This will, however, increase the flexibility of the system by having only those applications which are being used resident in program memory.

Work is also continuing on incorporating the real-time front end into the system. This development is just beginning at this time, but is not critical to DTW wafer testing since archived data will be used to initially test the system at real-time speeds.

b. Hardware

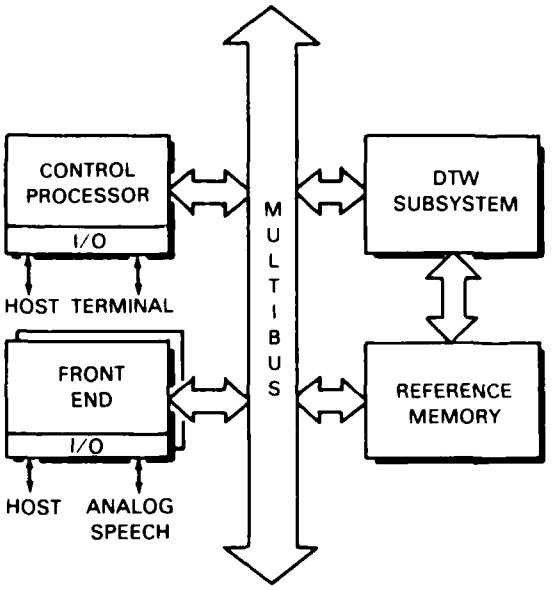
The Dynamic Time Warping (DTW) speech recognition system includes a wafer-scale array processor supported by (1) a front-end processor for performing feature extraction on input speech, (2) a subsystem containing the wafer and speech data memories, (3) a reference memory for storing a large vocabulary of speech templates, and (4) a general-purpose processor for high-level system control.

The five-board Multibus system is shown in Figure 6. High-level system control is exercised by a single board, 16-bit microcomputer acting as Multibus master. Front-end processing (two boards) includes sampling, A/D conversion, and a digital filter bank employing distributed single-chip DSP microcomputers. The DTW subsystem board hosts the DTW wafer and contains fast memories for storing input and level data, along with an 8 MIP microcontroller for wafer-level control. The fifth support system board is a 1-Mbyte reference memory for vocabulary storage.

Figure 7 shows the general wafer organization and its interface to the remainder of the speech recognition system, i.e., the DTW support system. The distance calculator (DCALC) operates on 6-bit values of input and reference coefficients. Each PCALC cell employs three channels of serial I/O for transferring path accumulation, column traceback, and reference word pointers as 24-bit values.

The system will operate at a 4-MHz clock rate, completing a new distance calculation and path update every 3 μs. A performance goal is the real-time processing of a 4000-word vocabulary with a 300-ms response time. The DTW support system, therefore, must (1) accept processed digitized speech from the front end and relay it and reference data to the wafer at 4 MHz; (2) accept 8 MHz serial wafer-level data, perform interlevel processing, and return serial-level data to the wafer; and (3) properly format recognition results for display at an operator terminal.

The control processor exercises high-level system control via Multibus, with access to the front end, DTW subsystem, and reference memory. Low-level control is embodied in the microcoded controller (μC) located on the subsystem board. As shown in Figure 7, the support system contains three dual-ported memories (level, input, reference) with interfaces both to Multibus and to the wafer. A fourth dual-ported memory is the μC's writable control store (WCS). Each of the three wafer memories has an autonomous fast address sequencer controlled by the μC. This microcontroller-based DTW subsystem with Multibus interface provides the flexibility to make control timing adjustments or to change memory contents throughout the wafer development process.



158027-N

Figure 6. DTW speech recognition system.

The development of support system hardware is essentially complete. It is now possible to run the front-end section and to apply all necessary signals at the wafer processor interface. Assembly language test routines have been written and verified to exercise all needed HK68 control processor capabilities. Similarly, a baseline microcontroller program has been used successfully at 8 MIPs for low-level wafer control.

The first planned use for the support system is to control a functional segment of the full wafer array comprised of individually packaged PCALC and DCALC cells. Delivery of these devices from the silicon foundry is expected shortly.

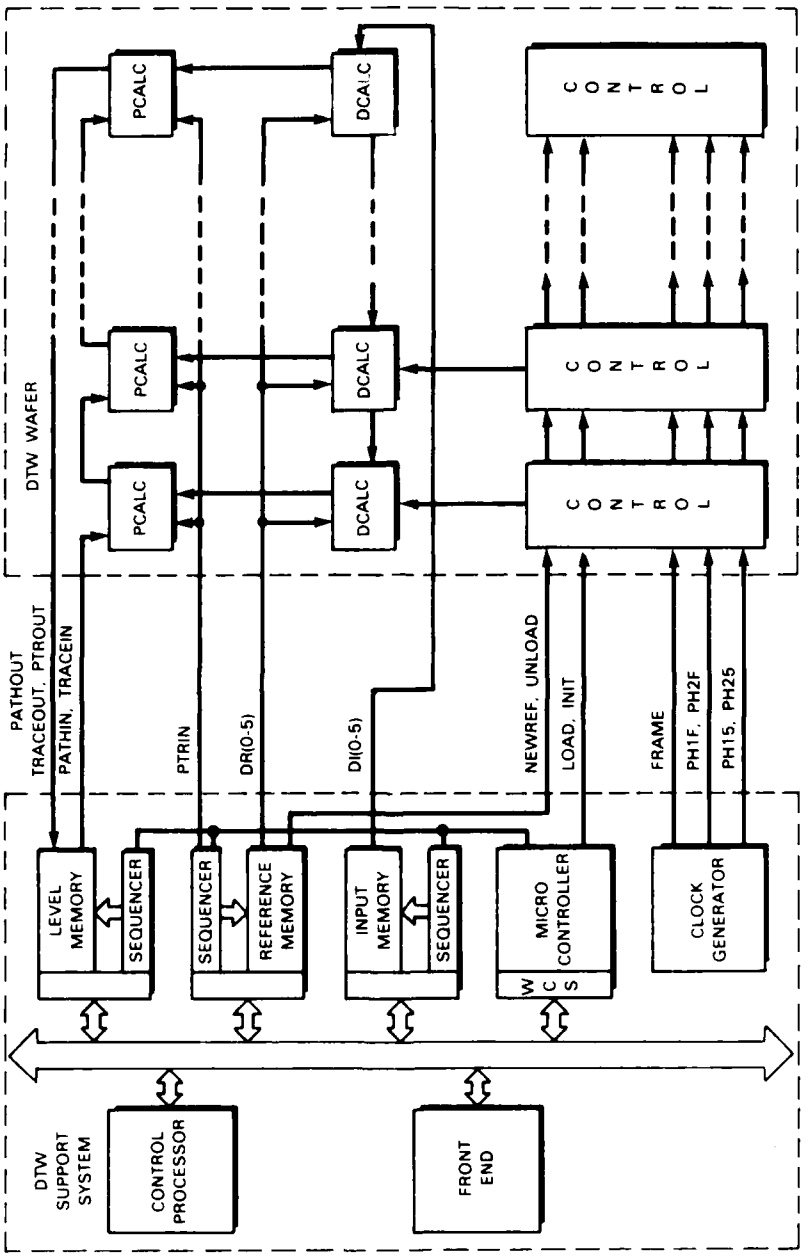


Figure 7. DTW support system/wafer interface.

158028-N

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER ESD-TR-85-293	2. GOVT ACCESSION NO. ADA165718
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) Restructurable VLSI Program	5. TYPE OF REPORT & PERIOD COVERED Semiannual Technical Summary 1 April — 30 September 1985
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gerald C. O'Leary	8. CONTRACT OR GRANT NUMBER(s) F19628-85-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173-0073	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order 3797 Program Element No.61101E Project No.3D30
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209	12. REPORT DATE 30 September 1985
	13. NUMBER OF PAGES 21
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Electronic Systems Division Hanscom AFB, MA 01731	15. SECURITY CLASS. (of this Report) Unclassified
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

VLSI	customization	systolic array
Restructurable VLSI (RVLSI)	hardware description language	integrator
programmable interconnect	placement	wafer-scale systems
defect avoidance	routing	speech recognition

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes work performed on the Restructurable VLSI Program sponsored by the Information Processing Techniques Office of the Defense Advanced Research Projects Agency during the period 1 April through 30 September 1985.